Direct Solvers for Sparse Matrices

X. Li *September 2006*

Direct solvers for sparse matrices involve much more complicated algorithms than for dense matrices. The main complication is due to the need for efficient handling the *fill-in* in the factors L and U. A typical sparse solver consists of four distinct steps as opposed to two in the dense case:

1. An ordering step that reorders the rows and columns such that the factors suffer little fill, or that the matrix has special structure such as block triangular form.
2. An analysis step or symbolic factorization that determines the nonzero structures of the factors and create suitable data structures for the factors.
3. Numerical factorization that computes the L and U factors.
4. A solve step that performs forward and back substitution using the factors.

There is a vast variety of algorithms associated with each step. The review papers by Duff [14] (see also [13, Chapter 6]) and Heath et al. [25] can serve as excellent reference of various algorithms. Usually steps 1 and 2 involve only the graphs of the matrices, and hence only integer operations. Steps 3 and 4 involve floating-point operations. Step 3 is usually the most time-consuming part, whereas step 4 is about an order of magnitude faster. The algorithm used in step 1 is quite independent of that used in step 3. But the algorithm in step 2 is often closely related to that of step 3. In a solver for the simplest systems, i.e., symmetric and positive definite systems, the four steps can be well separated. For the most general unsymmetric systems, the solver may combine steps 2 and 3 (e.g. SuperLU) or even combine steps 1, 2 and 3 (e.g. UMFPACK) so that the numerical values also play a role in determining the elimination order.

In the past 10 years, many new algorithms and software have emerged which exploit new architectural features, such as memory hierarchy and parallelism. In Table 1, we compose a rather comprehensive list of sparse direct solvers. It is most convenient to organize the software in three categories: the software for serial machines, the software for SMPs, and the software for distributed memory parallel machines.

Fair to say, there is no single algorithm or software that is best for all types of linear systems. Some software is targeted for special matrices such as symmetric and positive definite, some is targeted for the most general cases. This is reflected in column 3 of the table, "Scope". Even for the same scope, the software may decide to use a particular algorithm or implementation technique, which is better for certain applications but not for others. In column 2, "Technique", we give a high level algorithmic description. For a review of the distinctions between left-looking, right-looking, and multifrontal and their implications on performance, we refer the reader to the papers by Heath et al. [25] and Rothberg [31]. Sometimes the best (or only) software is not in public domain, but available commercially or in research prototypes. This is reflected this in column 4, "Contact", which could be the name of a company, or the name of the author of the research code.

In the context of shift-and-invert spectral transformation for eigensystem analysis, we need to factorize $A - \sigma I$, where A is fixed. Therefore, the nonzero structure of $A - \sigma I$ is fixed. Furthermore, for the same shift σ, it is common to solve many systems with the same matrix and different right-hand sides. (in which case the solve cost can be comparable to factorization cost.) It is reasonable to spend a little more time in steps 1 and 2 but speed up steps 3 and 4. That is, one can try different ordering schemes and estimate the costs of numerical factorization and solution based on symbolic factorization, and use the best ordering. For instance, in computing the SVD, one has

Code	Technique	Scope	Contact	
Serial platforms				
CHOLMOD	Left-looking	SPD	Davis	[8]
MA57	Multifrontal	Sym	HSL	[17]
MA41	Multifrontal	Sym-pat	HSL	[1]
MA42	Frontal	Unsym	HSL	[18]
MA67	Multifrontal	Sym	HSL	[15]
MA48	Right-looking	Unsym	HSL	[16]
Oblio	Left/right/Multifr.	sym, out-core	Dobrian	[12]
SPARSE	Right-looking	Unsym	Kundert	[27]
SPARSPAK	Left-looking	SPD, Unsym, QR	George et al.	[20]
SPOOLES	Left-looking	Sym, Sym-pat, QR	Ashcraft	[5]
SuperLLT	Left-looking	SPD	Ng	[30]
SuperLU	Left-looking	Unsym	Li	[10]
UMFPACK	Multifrontal	Unsym	Davis	[9]
Shared memory parallel machines				
BCSLIB-EXT	Multifrontal	Sym, Unsym, QR	Ashcraft et al.	[6]
Cholesky	Left-looking	SPD	Rothberg	[33]
DMF	Multifrontal	Sym	Lucas	[29]
MA41	Multifrontal	Sym-pat	HSL	[4]
MA49	Multifrontal	QR	HSL	[3]
PanelLLT	Left-looking	SPD	Ng	[23]
PARASPAR	Right-looking	Unsym	Zlatev	[34]
PARDISO	Left-right looking	Sym-pat	Schenk	[32]
SPOOLES	Left-looking	Sym, Sym-pat	Ashcraft	[5]
SuperLU_MT	Left-looking	Unsym	Li	[11]
TAUCS	Left/Multifr.	Sym, Unsym, out-core	Toledo	[7]
WSMP	Multifrontal	SPD, Unsym	Gupta	[24]
Distributed memory parallel machines				
DMF	Multifrontal	Sym	Lucas	[29]
DSCPACK	Multifrontal	SPD	Raghavan	[26]
MUMPS	Multifrontal	Sym, Sym-pat	Amestoy	[2]
PaStiX	Left-right looking*	SPD	CEA	[21]
PSPASES	Multifrontal	SPD	Gupta	[22]
SPOOLES	Left-looking	Sym, Sym-pat, QR	Ashcraft	[5]
SuperLU_DIST	Right-looking	Unsym	Li	[28]
S+	Right-looking†	Unsym	Yang	[19]
WSMP	Multifrontal	SPD, Unsym	Gupta	[24]

Table 1: Software to solve sparse linear systems using direct methods.

* In spite of the title of the paper

† Uses QR storage to statically accommodate any LU fill-in

Abbreviations used in the table:

SPD = symmetric and positive definite
Sym = symmetric and may be indefinite
Sym-pat = symmetric nonzero pattern but unsymmetric values
Unsym = unsymmetric
HSL = Harwell Subroutine Library: http://www.cse.clrc.ac.uk/Activity/HSL

the choice between shift-and-invert on AA^*, A^*A, and $\begin{bmatrix} 0 & A \\ A^* & 0 \end{bmatrix}$, all of which can have rather different factorization costs.

Some solvers have the ordering schemes built in, but others do not. It is also possible that the built-in ordering schemes are not the best for the target applications. It is sometimes better to substitute an external ordering scheme for the built-in one. Many solvers provide well-defined interfaces so that the user can make this substitution easily. One should read the solver documentation to see how to do this, as well as to find out the recommended ordering methods.

References

[1] P. R. Amestoy and I. S. Duff. Vectorization of a multiprocessor multifrontal code. *Int. J. of Supercomputer Applics.*, 3:41–59, 1989.

[2] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[3] P. R. Amestoy, I. S. Duff, and C. Puglisi. Multifrontal QR factorization in a multiprocessor environment. *Numer. Linear Algebra Appl.*, 3(4):275–300, 1996.

[4] Patrick R. Amestoy and Iain S. Duff. Memory management issues in sparse multifrontal methods on multiprocessors. *Int. J. Supercomputer Applics*, 7:64–82, 1993.

[5] C. Ashcraft and R. Grimes. SPOOLES: An object-oriented sparse matrix library. In *Proceedings of the Ninth SIAM Conference on Parallel Processing*, 1999. (`http://www.netlib.org/linalg/spooles`).

[6] BASLIB-EXT: Sparse matrix software. `http://www.boeing.com/phantom/bcslib-ext/`.

[7] D. Chen, V. Rotkin, and S. Toledo. TAUCS: A Library of Sparse Linear Solvers, Tel-Aviv Univesity. (`http://www.tau.ac.il/~stoledo/taucs/`).

[8] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 8xx: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. Technical Report TR-2006-005, University of Florida, 2006. (`http://www.cise.ufl.edu/research/sparse/cholmod/`).

[9] T. A. Davis. Algorithm 832: UMFPACK V4.3, an unsymmetric-pattern multifrontal method with a column pre-ordering strategy. *ACM Trans. Mathematical Software*, 30(2):196–199, June 2004. (`http://www.cise.ufl.edu/research/sparse/umfpack/`).

[10] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.*, 20(3):720–755, 1999. (`http://crd.lbl.gov/~xiaoye/SuperLU`).

[11] James W. Demmel, John R. Gilbert, and Xiaoye S. Li. An asynchronous parallel supernodal algorithm for sparse Gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 20(4):915–952, 1999. (`http://crd.lbl.gov/~xiaoye/SuperLU`).

[12] F. Dobrian and A. Pothen. Oblio: a sparse direct solver library for serial and parallel computations. Technical report, Old Dominion University, 2000.

[13] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers.* SIAM, Philadelphia, PA, 1998.

[14] Iain S. Duff. Direct methods. Technical Report RAL-98-056, Rutherford Appleton Laboratory, 1998.

[15] I.S Duff and J. K. Reid. MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems. Technical Report RAL-95-001, Rutherford Appleton Laboratory, 1995.

[16] I.S Duff and J. K. Reid. The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations. *ACM Trans. Math. Software*, 22:187–226, 1996.

[17] I.S Duff and J.K Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software*, 9(3):302–325, September 1983.

[18] I.S Duff and J. A. Scott. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Math. Software*, 22(1):30–45, 1996.

[19] Cong Fu, Xiangmin Jiao, and Tao Yang. Efficient sparse LU factorization with partial pivoting on distributed memory architectures. *IEEE Trans. Parallel and Distributed Systems*, 9(2):109–125, 1998. (`http://www.cs.ucsb.edu/research/S+`).

[20] A. George and J. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981. (jageorge@sparse1.uwaterloo.ca).

[21] D. Goudin, P. Henon, F. Pellegrini, P. Ramet, and J. Roman. Parallel Sparse matriX package, LaBRI, Université Bordeaux I, Talence, France. (`http://www.labri.fr/Perso/~ramet/pastix/`).

[22] A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Trans. Parallel and Distributed Systems*, 8:502–520, 1997. (`http://www.cs.umn.edu/~mjoshi/pspases`).

[23] A. Gupta, E. Rothberg, E. Ng, and B. W. Peyton. Parallel sparse Cholesky factorization algorithms for shared-memory multiprocessor systems. In R. Vichnevetsky, D. Knight, and G. Richter, editors, *Advances in Computer Methods for Partial Differential Equations–VII*, pages 622–628. IMACS, 1992. (egng@lbl.gov).

[24] Anshul Gupta. WSMP: Watson Sparse Matrix Package. IBM T.J. Watson Research Center, Yorktown Heights. (`http://www-users.cs.umn.edu/~agupta/wsmp.html`).

[25] M. Heath, E. Ng, and B. Peyton. Parallel algorithms for sparse linear systems. *SIAM Review*, 33:420–460, 1991.

[26] Michael T. Heath and Padma Raghavan. Performance of a fully parallel sparse solver. *Int. J. Supercomputer Applications*, 11(1):49–64, 1997. `http://www.cse.psu.edu/~raghavan`.

[27] Kenneth Kundert. Sparse matrix techniques. In Albert Ruehli, editor, *Circuit Analysis, Simulation and Design*. North-Holland, 1986. (http://www.netlib.org/sparse).

[28] Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110–140, June 2003.

[29] Robert Lucas. Private communication (rflucas@lbl.gov), 2000.

[30] Esmond G. Ng and Barry W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comput.*, 14(5):1034–1056, September 1993. (egng@lbl.gov).

[31] E. Rothberg. *Exploiting the memory hierarchy in sequential and parallel sparse Cholesky factorization.* PhD thesis, Dept. of Computer Science, Stanford University, December 1992.

[32] O. Schenk, K. Gärtner, and W. Fichtner. Efficient sparse LU factorization with left–right looking strategy on shared memory multiprocessors. *BIT*, 40(1):158–176, 2000.

[33] J.P. Singh, W-D. Webber, and A. Gupta. Splash: Stanford parallel applications for shared-memory. *Computer Architecture News*, 20(1):5–44, 1992. (`http://www-flash.stanford.edu/apps/SPLASH`).

[34] Z. Zlatev, J. Waśniewski, P. C. Hansen, and Tz. Ostromsky. PARASPAR: a package for the solution of large linear algebraic equations on parallel computers with shared memory. Technical Report 95-10, Technical University of Denmark, September 1995.